**Database: Diagramma E/R – Schema Logico**

Quando si è passati all’analisi della gestione dei dati presenti nel sistema, come già citato, è subito emerso come i diversi dati fossero strettamente legati tra loro tramite relazioni, che creavano così il “pacchetto” di informazioni del singolo paziente registrato e gestito dal sistema. Finita questa fase si è passati alla creazione del Diagramma E/R (Entity Relationship), stilando così un primo schema che rappresentava come avrebbe dovuto essere costruito il sistema di storage sulla base delle considerazioni effettuate precedentemente. Una volta corrette alcune imprecisioni, e confermato la bozza si è passati alla creazione del diagramma E/R ristrutturato, riportato nella figura seguente:

*(FIGURA ER)*

Come si può notare *Paziente* è collegato tramite una relazione 1-N sia con *Somministrazioni,* che con *Ricovero*. A primo impatto si nota subito come la prima non sia necessaria in quanto, tramite le diverse relazioni, si riesce a collegare perfettamente il paziente con le sue somministrazioni (*Paziente* -> *Ricovero* -> *Prescrizioni* -> *Somministrazioni*). Si è però deciso di aggiungere questa relazione tra le due entità in quanto spesso, all’interno del software, viene richiesta l’informazione di un determinato *Paziente* e le sue *Somministrazioni*, senza che sia necessario avere informazioni relative al *Ricovero* e/o *Prescrizioni*. Il team ha deciso di evitare tre JOIN concatenati, fornendo più semplicemente un rifermento al *Paziente* all’interno della *Somministrazione*, in modo tale da ottenere le informazioni necessarie con il minimo uso di risorse.

Successivamente è stato creato lo schema Logico sulla base dello schema E/R, rispettando le regole di transizione tra i due schemi. Una volta controllate le forme normali dello schema, si è proseguito con l’implementazione del Database in modo tale che le diverse tabelle rispettassero quanto riportato all’interno dello schema.

(*FIGURA LOGICO)*

**Data Generator**

Per quanto riguarda la generazione dei dati relativi al monitoraggio dei diversi pazienti ricoverati, è stato scelto di creare dei processi attivi in background dedicati a questo compito. Nel software sono presenti tre diversi generatori, uno per ogni gruppo di dati (battito cardiaco, pressione sanguinea, temperatura). Per l’implementazione si è optato di utilizzare delle distribuzioni Gaussiane, dove la media e varianza è stata calibrata rispetto ai dati segnalati dalla Dott.ssa De Carli durante l’incontro.   
I generatori, a regime normale, generano dati all’interno di intervalli accettabili per i segnali vitali di un paziente ricoverato in terapia intensiva. Una volta che viene richiesta la generazione di un allarme i generatori evolvono cambiando media e varianza della distribuzione, in questo modo verranno generati dati che non rispettano gli standard di valori accettabili. I valori rimangono fuori dalla norma fintanto che l’allarme è attivo, una volta disattivato o scaduto viene traslata la curva gaussiana entro i range di dati accettabili.

**MVC INTRO**

La scelta del design architetturale è ricaduta sul modello MVC, Model View Controller. La scelta è stata guidata e ispirata dai sistemi web dove è presente un gestore centralizzato del software che gestisce le varie componenti e la loro comunicazione con il sistema di storage. Seguendo questa filosofia è stato sviluppato SICURA: un sistema centralizzato che controlla i diversi component, i segnali generati dai loro controllori, il loro accesso con lo stato attuale del sistema e gestisce la coerenza e update del database, sulla base dei dati che vengono man mano generati dall’applicativo e caricati nello stato.

**MVC SYSTEM**

Il software ha una classe singleton Sistema, la quale rappresenta il core del programma, che gestisce le interfacce del sistema, lo stato centralizzato e fa da ponta con il database.

Durante il bootstrapping del software viene creata l’istanza della classe Sistema; in questo passaggio la classe va a settare lo stato iniziare dei diversi elementi che ne sono alla base e permettono l’esecuzione del programma. Nello specifico quando si va a creare l’unica istanza vengono eseguiti diversi passaggi rispettando l’ordine specificato nel sequence diagram qui riportato.

(*SEQUENCE DIAGRAM BOOTSTRAPPING*)

In un primo momento viene creata un’istanza di Store, al quale vengono associati gli handler ai diversi comandi di update dello stato, e una di InterfacesController, al quale verrà dato il compito di gestire l’evoluzione delle interfacce durante la vita del software. Una volta finita questa fase viene inviato il comando di LOAD, quest’ultimo viene gestito dallo Store caricando da database tutti i dati in esso contenuti e inizializzando cosi lo State iniziale del software. Ultimo, non per importanza, viene popolata la pool di interfacce gestite da InterfacesController, associando una stringa a una determinata interfaccia. Tramite questo controllore il sistema può decidere a seconda della sua evoluzione quale interfaccia caricare semplicemente inviando un messaggio di update.

Le interfacce come anche i controllori, per quanto riguarda i tre diversi utenti all’interno del sistema, vengono istanziate e ottenute tramite le sei factory presenti all’interno del sistema. Infatti nel software sono presenti diverse factory, tre dedicate al caricamento dei component e altrettante dedicate al caricamento dei controllori delle interfacce, seguendo la seguente logica.

(*INTERFACES CONTROLLER FACTORY*)

Terminato questo processo il Sistema è stato correttamente istanziato e portato allo stato iniziale dal quale potrà poi evolvere a seconda delle richieste da parte dell’utente.

**COMPONENT - CONTROLLER INTERCOMUNICATION**

Di seguito l’analisi di come un qualsiasi component comunica con il suo controller a seconda degli eventi generati da azioni da parte dello user.

(*SEQUENCE DIAGRAM USER INTERACION*)

I vari componenti dell’interfaccia sono mappati tramite degli id dal controllore. Quando lo user interagisce con l’interfaccia attiva un determinato componente che invoca il suo EventHandler. Quest’ultimo una volta aggiorna internamente l’interfaccia evolvendo eventuali campi e informazioni, e successivamente se necessario invia un comando di update allo Store che gli era stato iniettato durante la fase di creazione del component. Il controller dell’interfaccia rimane poi in ascolto di eventuali messaggi di update dello State e/o utilizzi degli elementi presenti nell’interfaccia, che nel caso in cui vanno a variare i dati correntemente attivi e salvati, aggiorna di conseguenza l’interfaccia mostrando all’utente, tramite il pattern Observer, un update in tempo reale.